



# Approximate evaluation and voltage assignment for order/degree problem

Ibuki Kawamata

CANDAR2017, Graph Golf workshop

22 Nov. 2017

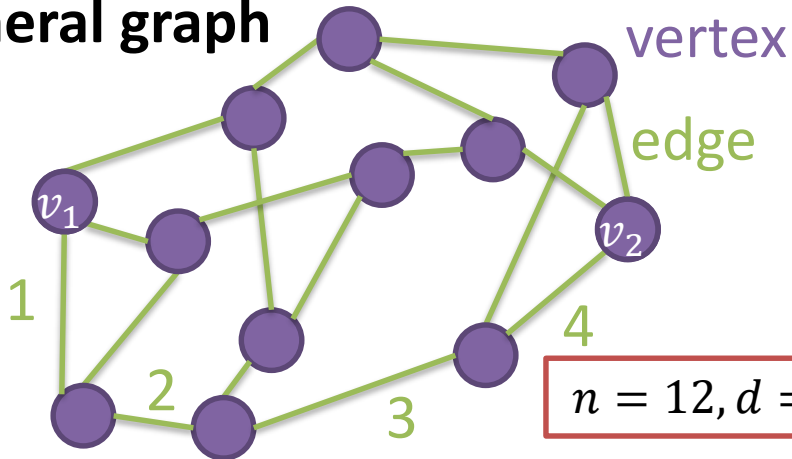
[kawamata@molbot.mech.tohoku.ac.jp](mailto:kawamata@molbot.mech.tohoku.ac.jp)

# Graph Golf

- Last year, I attended CANDAR 2016 in another workshop, and noticed the Graph Golf competition
- **Graph Golf (general)**
  - *“The order/degree problem with parameters  $n$  and  $d$  : Find a graph with minimum diameter over all undirected graphs with the number of vertices =  $n$  and degree  $\leq d$ ”*
- **Graph Golf (grid)**
  - *“The order/degree problem on a grid graph with a limited edge length  $r$ : Do the same as above, but on a  $\sqrt{n} \times \sqrt{n}$  square grid in a two-dimensional Euclidean space, keeping the lengths of the edges  $\leq r$  in Manhattan distance.”*

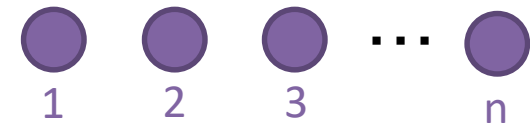
# Undirected unweighted graph

General graph

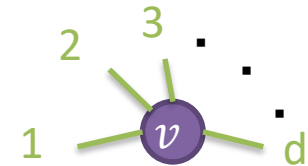


Graph :  $G = (V, E)$

Order:  $n$



Degree :  $d$



Shortest path length :  $s(v_1, v_2)$

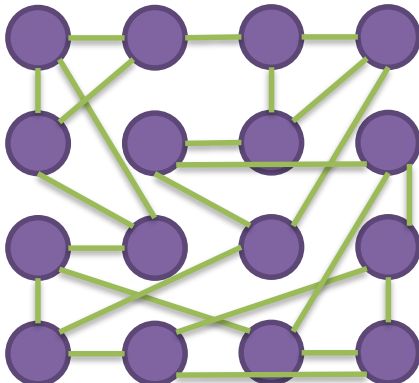
Diameter :  $k = \max\{s(v_1, v_2) | v_1, v_2 \in V, v_1 \neq v_2\}$

Average shortest path length :

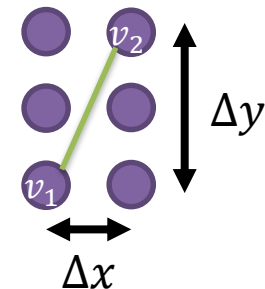
$ASPL = \text{average}\{s(v_1, v_2) | v_1, v_2 \in V, v_1 \neq v_2\}$

Grid graph

$n = 16, d = 3, r = 3$



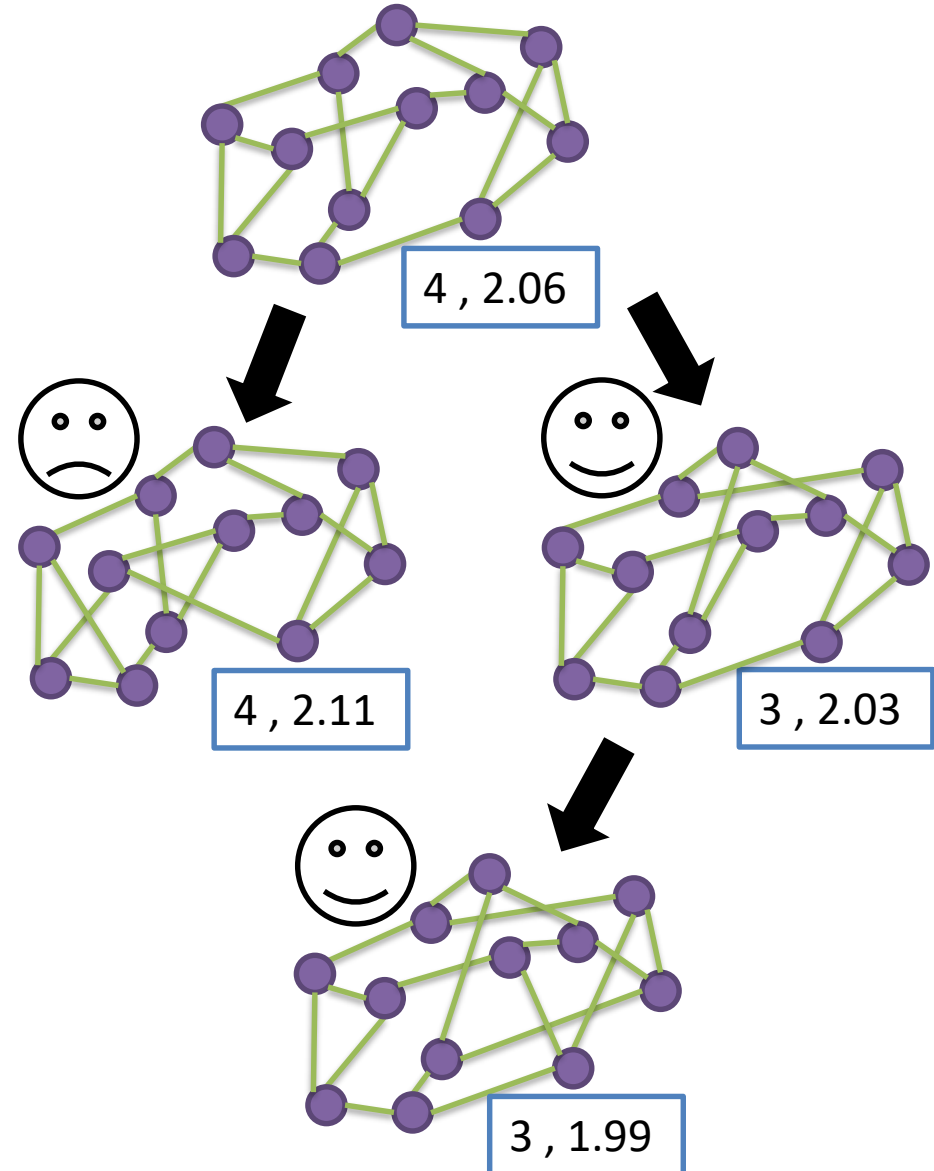
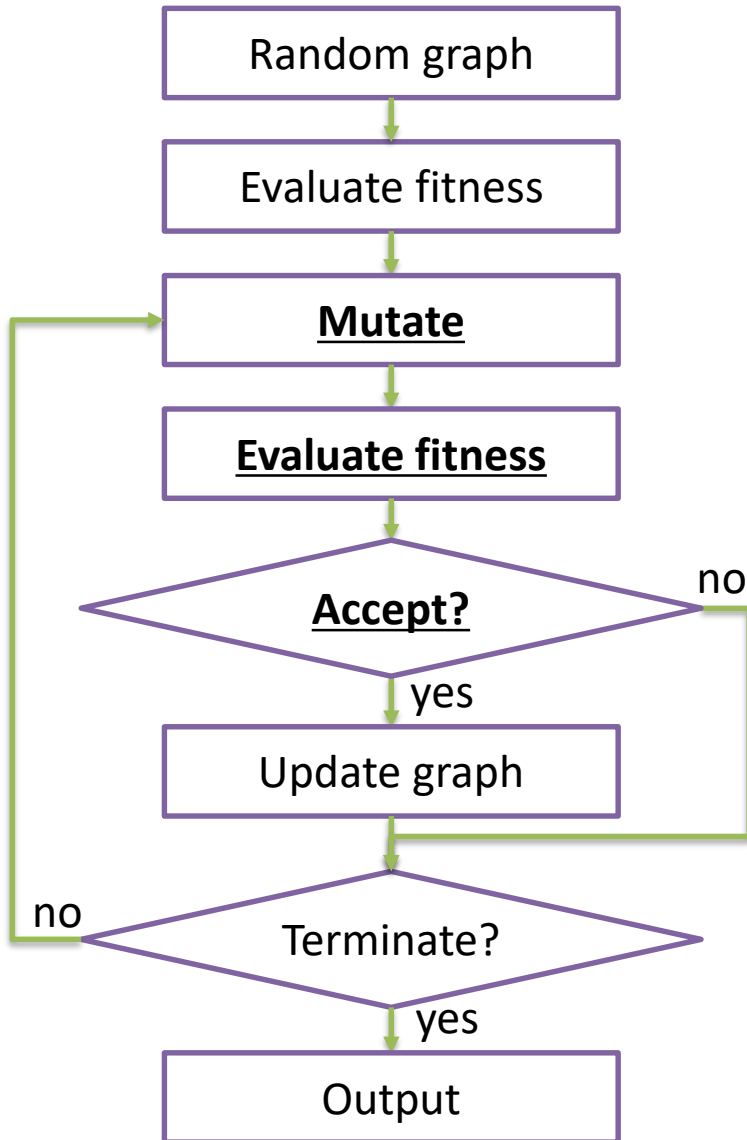
Edge length :  $r(v_1, v_2) = \Delta x + \Delta y$



# Strategy

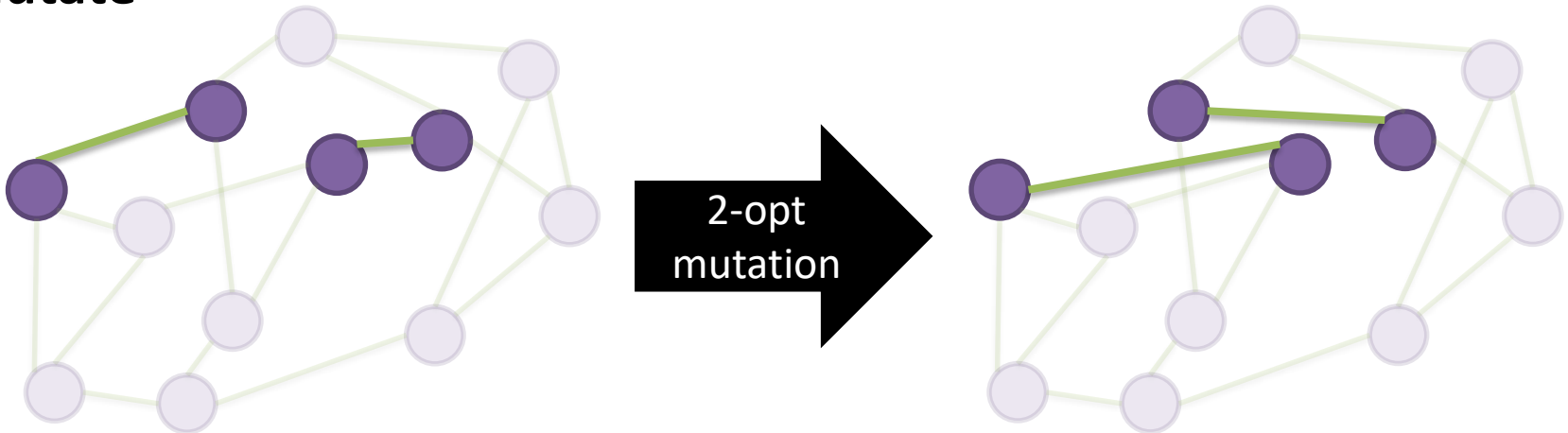
- How do we search for a good graph?
  - Largest  $n$  is 100,000 in general graph and 10,000 in grid graph, which results in a huge search space
  - Want to avoid algorithms that require  $O(n^2)$  time and memory
- I used simulated annealing algorithm
  - For grid graph, accelerate the evaluation by introducing **approximation**
  - For general graph, limit to **voltage graph**

# Simulated annealing algorithm



# 2-opt mutation, accept probability

**Mutate**



**Accept?**

$$\text{Accept probability} : \begin{cases} 1 & \text{if } f_2 - f_1 < 0 \\ \exp\left(-\frac{f_2 - f_1}{t}\right) & \text{otherwise} \end{cases}$$

$f_1$  and  $f_2$  are the fitness values before and after the mutation  
 $t$  is a temperature parameter

Computing  $\Delta f = f_2 - f_1$  is enough to make the decision

# Evaluating fitness

## Evaluate fitness (exact version)

Fitness value :  $f = w \times k + ASPL$

$$w = \begin{cases} 10 & \text{general graph} \\ 100 & \text{grid graph} \end{cases}$$

$k$  and  $ASPL$  are diameter and average shortest path length

- Naïve way to compute  $k$  and  $ASPL$  is by running **breadth first search (BFS)** from each vertex
  - This is  $O(n^2 \times e)$  time algorithm
- Is there any better way to compute  $f$  (or  $\Delta f$  of 2-opt mutation)?
- Since most of the graph are preserved, it is not necessary to compute all-to-all shortest path length
- I first tried to find a **set of critical vertex** that would update the shortest path length by 2-opt mutation

# BFS

BFS( $v$ )

depth[ $v$ ] := 0

Queue.enqueue( $v$ )

**while** Queue.size() > 0

$v_1$  := Queue.head()

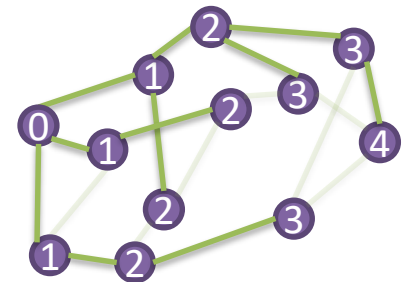
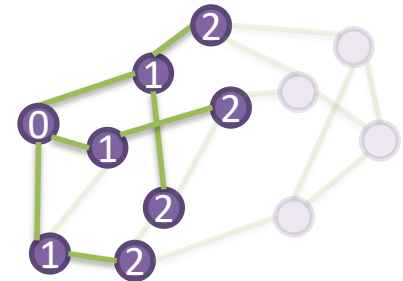
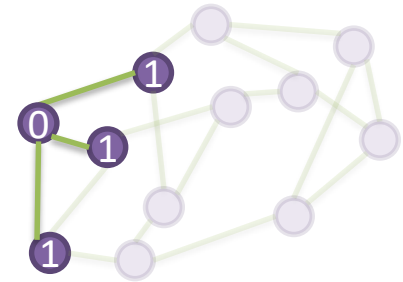
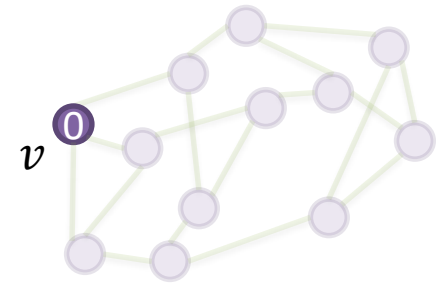
**for each**  $v_2$  :=  $v_1$ .neighbor()

**if** depth[ $v_2$ ] = **undefined**

            depth[ $v_2$ ] := depth[ $v_1$ ] + 1

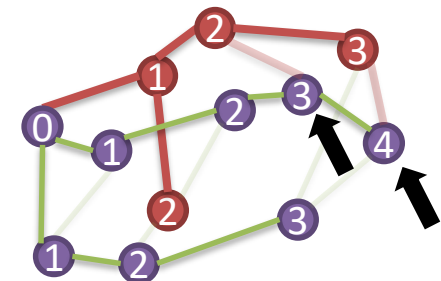
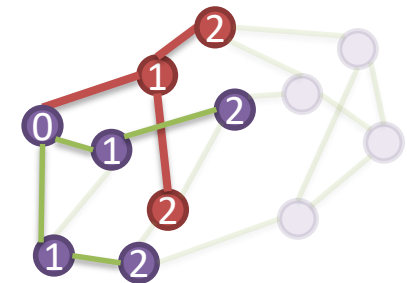
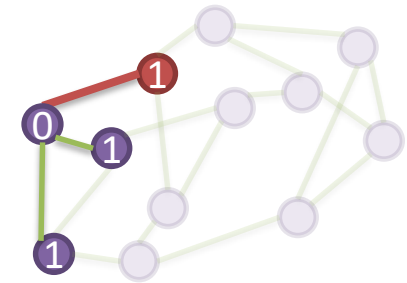
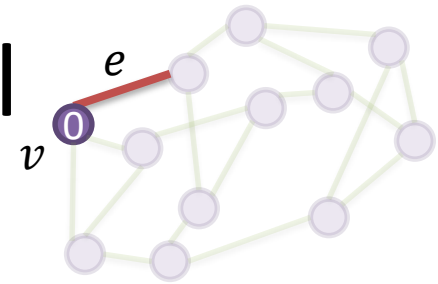
            Queue.enqueue( $v_2$ )

    Queue.dequeue()





# (simple) critical vertex set



CRITICAL'(v, e)

depth[v] := 0

Queue.enqueue(v)

v' := e.opposite(v)

critical := {v'}

**while** Queue.size() > 0

    v<sub>1</sub> := Queue.head()

**for each** v<sub>2</sub> := v<sub>1</sub>.neighbor()

**if** depth[v<sub>2</sub>] = **undefined**

            depth[v<sub>2</sub>] := depth[v<sub>1</sub>] + 1

            Queue.enqueue(v<sub>2</sub>)

**if** v<sub>1</sub> ∈ critical **then** critical := critical ∪ {v<sub>2</sub>}

**else if** depth[v<sub>2</sub>] = depth[v<sub>1</sub>] + 1 **and** v<sub>2</sub> ∈ critical **and** v<sub>1</sub> ∉ critical

critical := critical \ {v<sub>2</sub>}

    Queue.dequeue()

**return** critical

# Critical vertex set

CRITICAL( $v, e_1, e_2$ )

depth[ $v$ ] := 0

Queue.enqueue( $v$ )

$v' := e_1.opposite(v)$

critical := { $v'$ }

**while** Queue.size() > 0

$v_1 :=$  Queue.head()

**for each**  $v_2 := v_1.neighbor()$

**if** ( $v_1, v_2$ )  $\neq e_2$

**if** depth[ $v_2$ ] = **undefined**

        depth[ $v_2$ ] := depth[ $v_1$ ] + 1

        Queue.enqueue( $v_2$ )

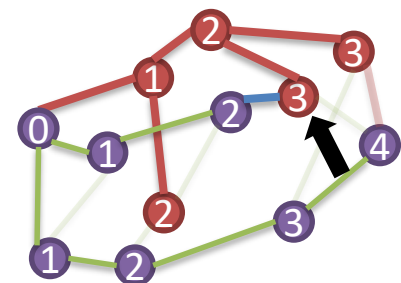
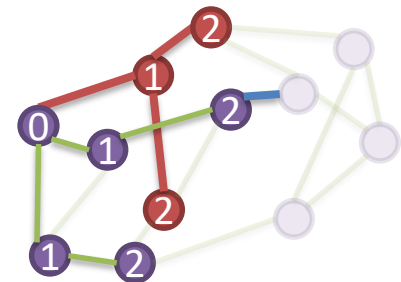
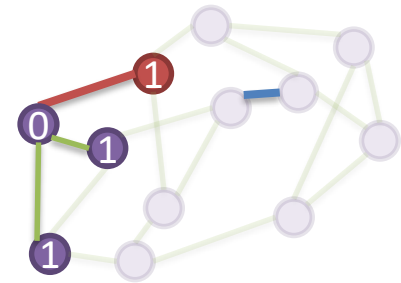
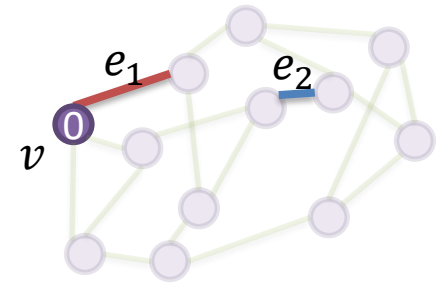
**if**  $v_1 \in$  critical **then** critical := critical  $\cup$  { $v_2$ }

**else if** depth[ $v_2$ ] = depth[ $v_1$ ] + 1 **and**  $v_2 \in$  critical **and**  $v_1 \notin$  critical

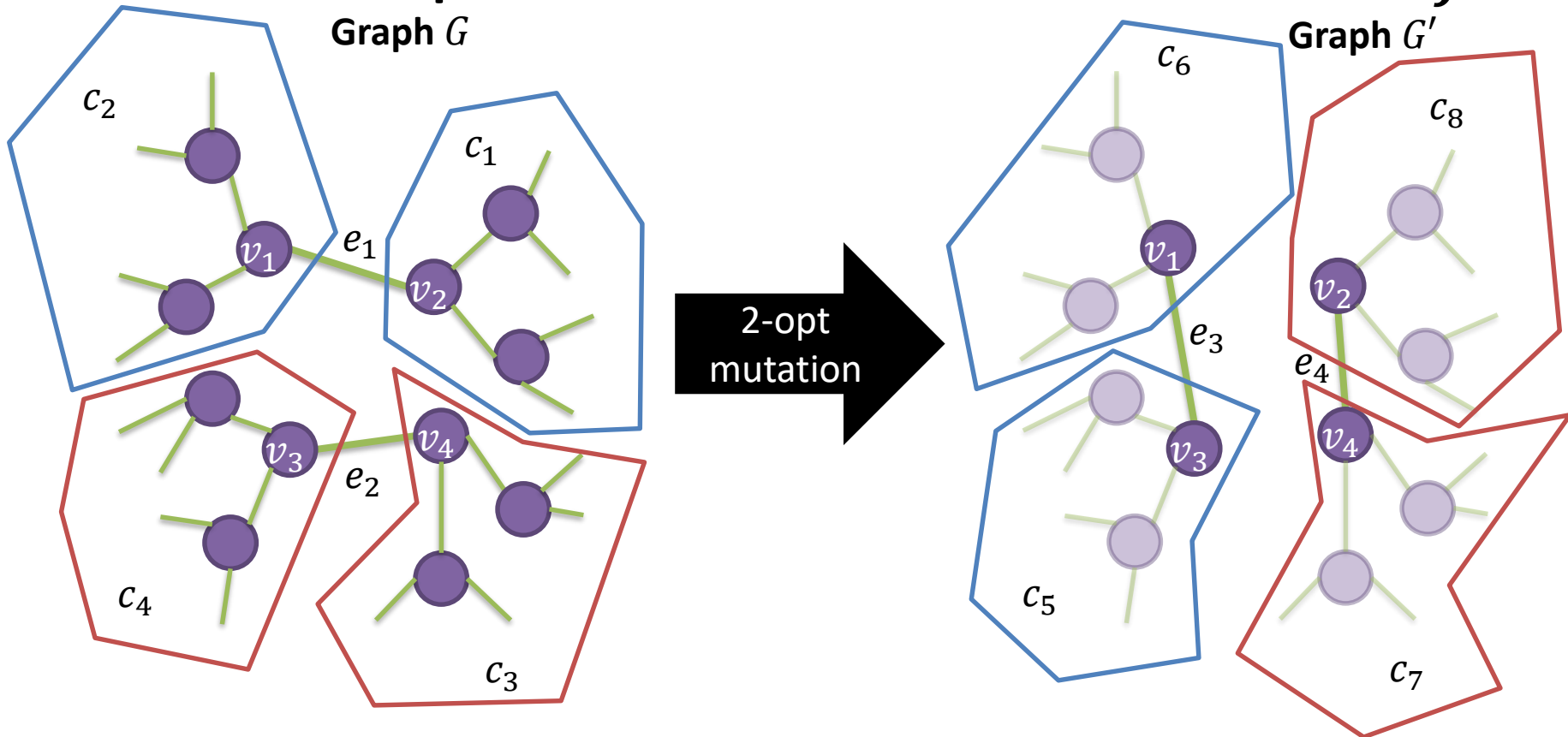
          critical := critical  $\setminus$  { $v_2$ }

    Queue.dequeue()

**return** critical



# Potential pairs of vertex that affect $\Delta f$



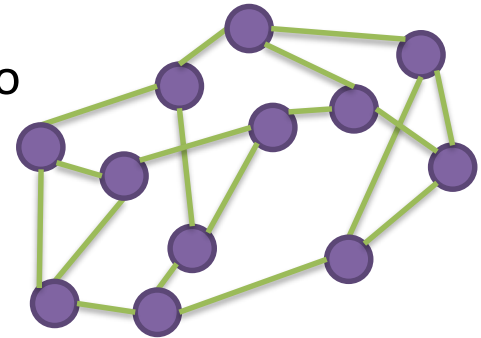
$$\begin{aligned} c_1 &= G.\text{CRITICAL}(v_1, e_1, e_2) \\ c_2 &= G.\text{CRITICAL}(v_2, e_1, e_2) \\ c_3 &= G.\text{CRITICAL}(v_3, e_2, e_1) \\ c_4 &= G.\text{CRITICAL}(v_4, e_2, e_1) \end{aligned}$$

$$\begin{aligned} c_5 &= G'.\text{CRITICAL}(v_1, e_3, e_4) \\ c_6 &= G'.\text{CRITICAL}(v_3, e_3, e_4) \\ c_7 &= G'.\text{CRITICAL}(v_2, e_4, e_3) \\ c_8 &= G'.\text{CRITICAL}(v_4, e_4, e_3) \end{aligned}$$

$$\text{pairs} = c_1 \otimes c_2 \cup c_3 \otimes c_4 \cup c_5 \otimes c_6 \cup c_7 \otimes c_8$$

# Computing exact $\Delta f$

- To compute  $\Delta f$  using the potential pair, it is necessary to prepare the **occurrence of each shortest path length**
  - occurrence[0] =  $n$
  - occurrence[1] =  $\frac{d \times n}{2}$  (if regular)
  - . . .
  - occurrence[ $i$ ] =  $|\{(v_1, v_2) | v_1, v_2 \in V, s(v_1, v_2) = i\}|/2$
- $k$  and  $ASPL$  are computed from the occurrence list
- By 2-opt mutation, the occurrence is updated by using BFS
  - Can be efficiently updated by just considering the potential pairs



occurrence[0] = 12  
 occurrence[1] = 18  
 occurrence[2] = 26  
 occurrence[3] = 19  
 occurrence[4] = 3

UPDATE(occurrence, pairs,  $G, G'$ )

**for each**  $(v_x, v_y) \in \text{pairs}$

$\text{bfs}_1 := G.\text{BFS}(v_x)$

$\text{occurrence}[\text{bfs}_1.\text{depth}[v_y]] := \text{occurrence}[\text{bfs}_1.\text{depth}[v_y]] - 1$

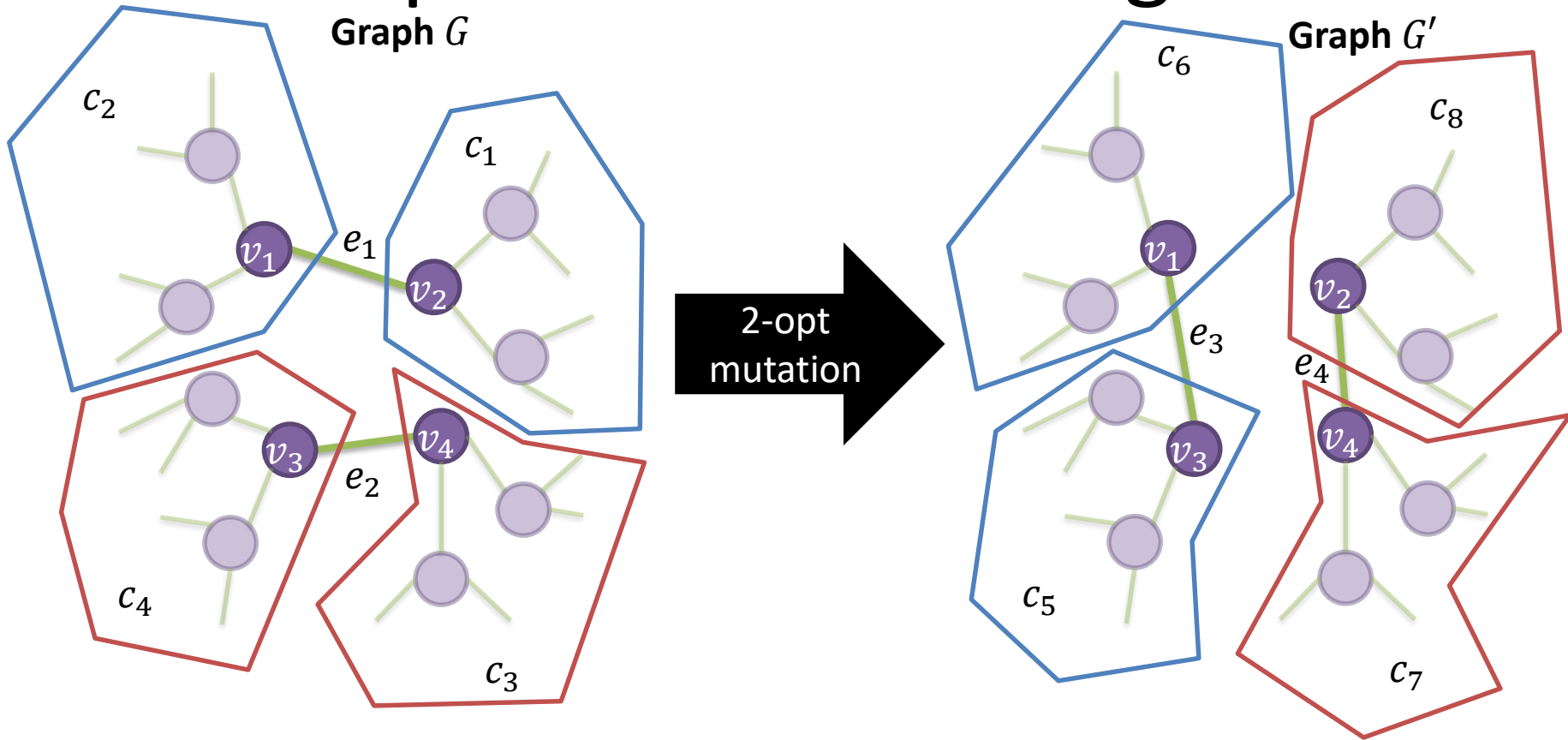
$\text{bfs}_2 := G'.\text{BFS}(v_x)$

$\text{occurrence}[\text{bfs}_2.\text{depth}[v_y]] := \text{occurrence}[\text{bfs}_2.\text{depth}[v_y]] + 1$

# From exact to approximate $\Delta f$

- Some optimization was possible in BFS during the UPDATE
  - Halting the search if there are no more critical vertex
  - Ignoring a pair of vertex that has a path length longer than the diameter when  $e_1$  or  $e_2$  is used
  - However, the algorithm to compute exact  $\Delta f$  still requires  $O(n^2 \times e)$  time
  - Could not get graphs better than those in the ranking page
- I decide to use another approximate  $\Delta f$  that requires  $O(n \times e)$  time algorithm

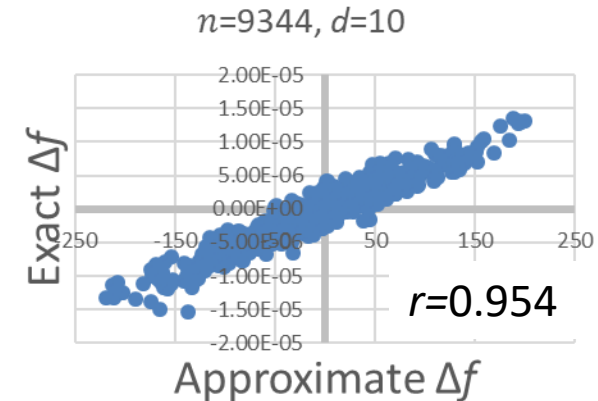
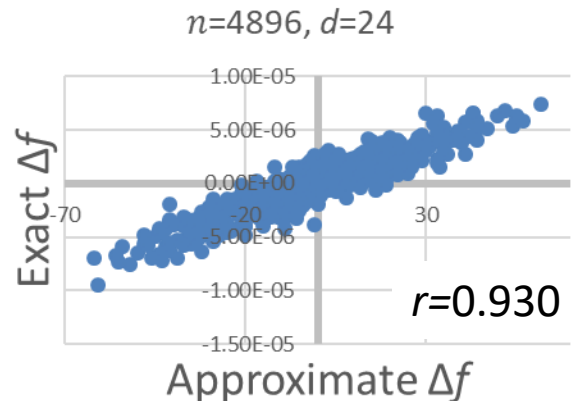
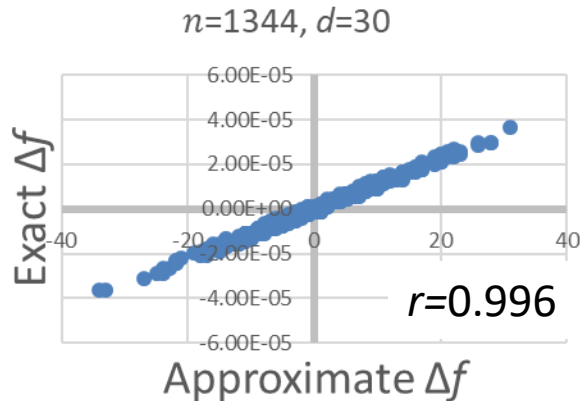
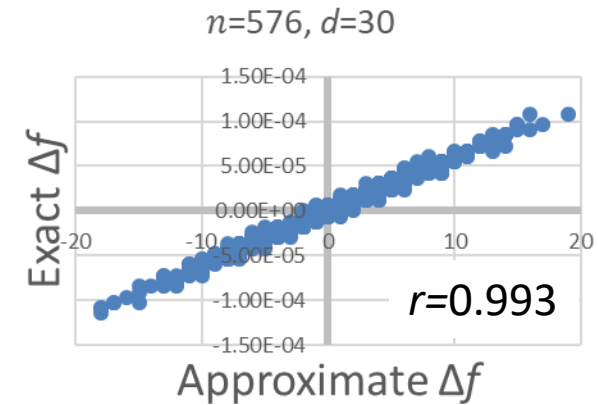
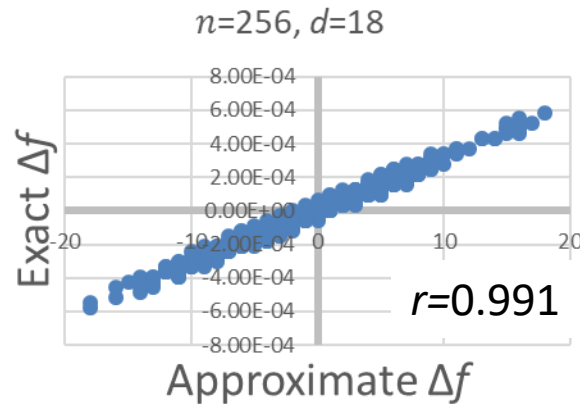
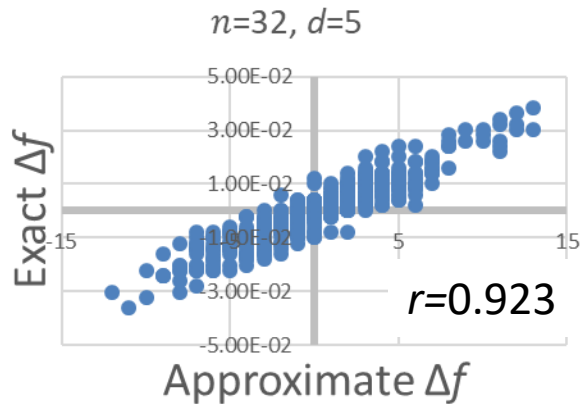
# Importance of an edge?



Can we use the number of vertex in  $c_1$  and  $c_2$  to estimate how  $e_1$  is important? (the bigger, the better)

$$\Delta f_{\text{approximate}} = |c_1| + |c_2| + |c_3| + |c_4| - |c_5| - |c_6| - |c_7| - |c_8|$$

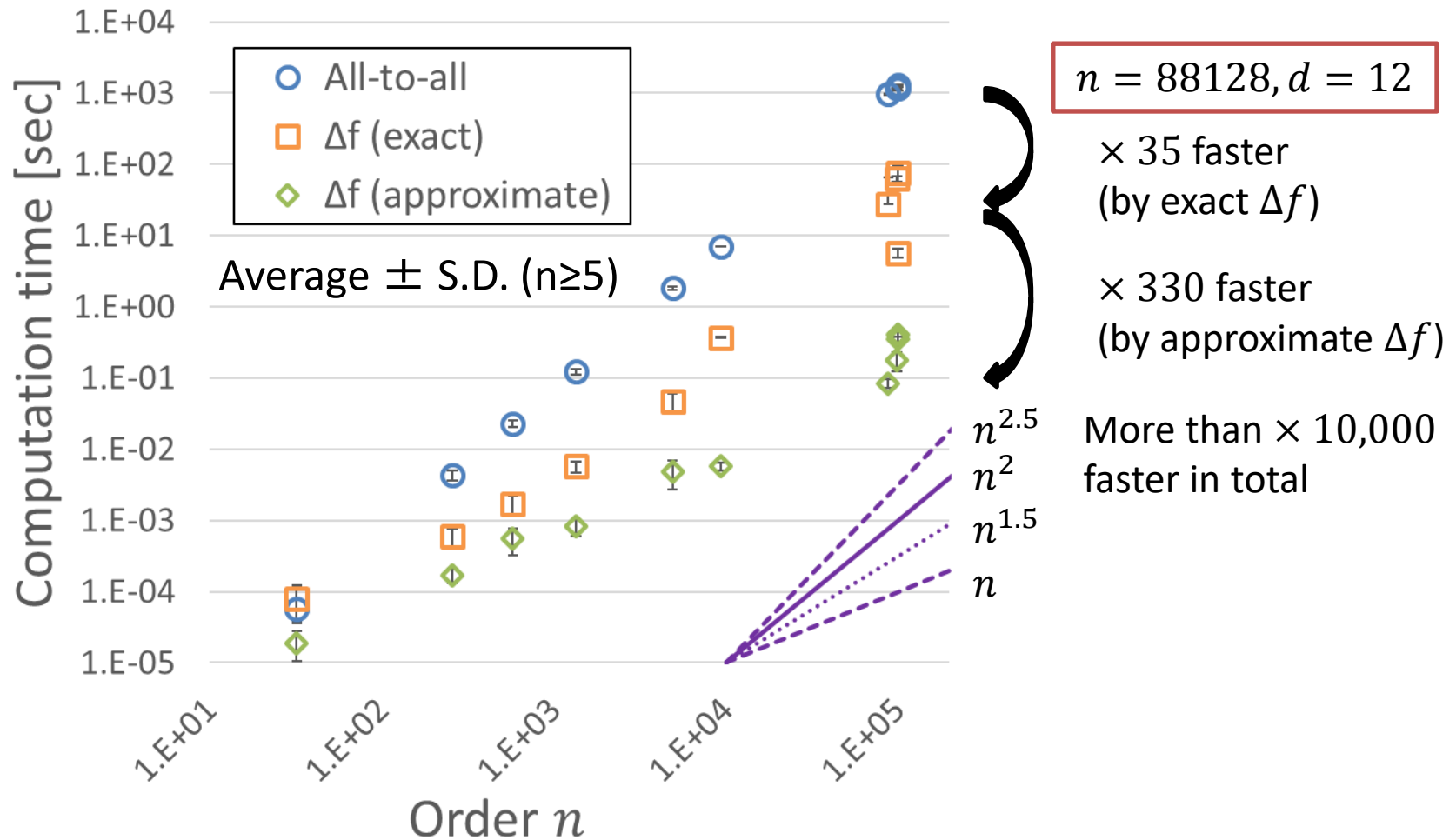
# Approximate and exact $\Delta f$



Correlation coefficient are from the distribution of 500 random 2-opt mutation

Approximate  $\Delta f$  was highly related to exact  $\Delta f$

# Computation time for evaluation



Accelerate the evaluation (scaling change from  $n^2$  to  $n$ )



# Result (grid graph)

- Run the simulated annealing algorithm using the approximate  $\Delta f$  for about 24 hours each
  - Submit the best graphs below in Graph Golf 2017

Order $n$	Degree $d$	Length $r$	Diameter $k$	$ASPL$	$ASPL$ (second best)
10,000	3	18	17	11.4	11.5
10,000	3	33	16	11.2	11.3
10,000	9	6	33	11.59	11.61
10,000	9	18	11	5.15	5.25
10,000	9	33	7	4.56	4.61
10,000	28	6	33	11.5	11.6
10,000	28	18	11	4.39	4.49
10,000	28	33	6	3.26	3.32

For  $n = 256$ , my results were not so good (after 2 hours search)

# Applying to general graph?

- Using the same approach to the general graph was not good
- I employ another mathematical approach invented in degree/diameter problem
  - Most of the known solutions (orange) are constructed using the **voltage graph**
  - The approach is also suitable to the order/degree problem

Table of the orders of the largest known graphs for the undirected degree diameter problem

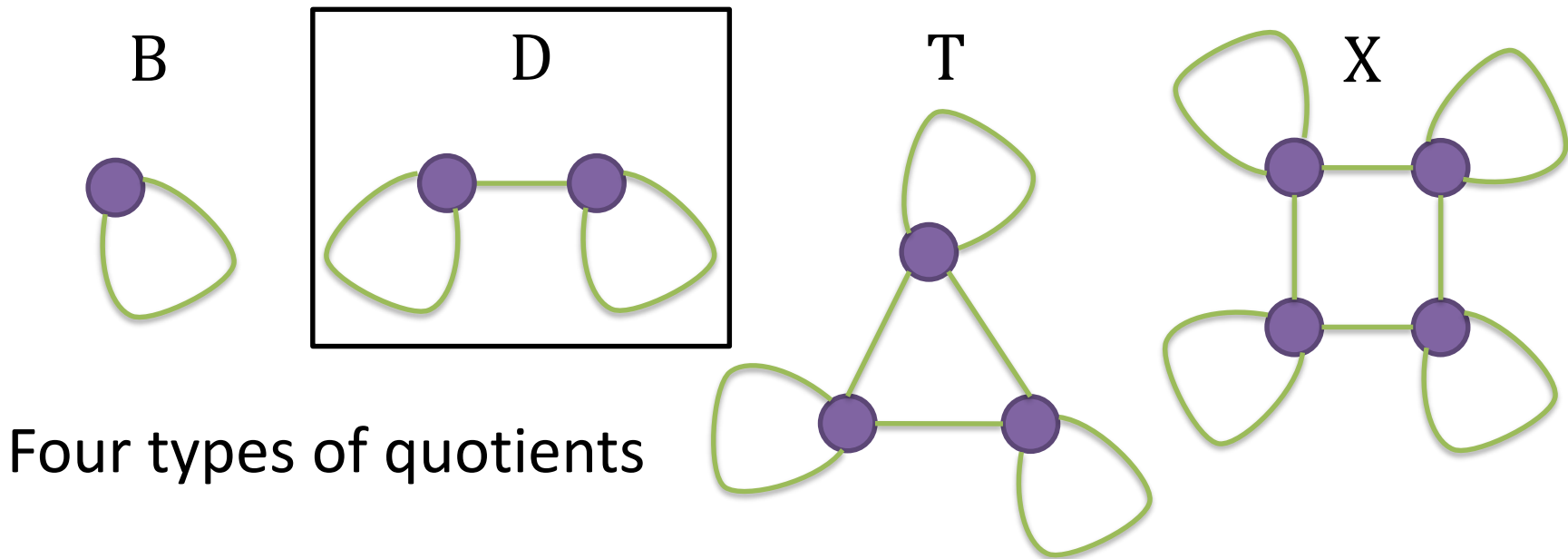
$d \backslash k$	2	3	4	5	6	7	8	9	10
3	10	20	38	70	132	196	336	600	1 250
4	15	41	98	364	740	1 320	3 243	7 575	17 703
5	24	72	212	624	2 772	5 516	17 030	57 840	187 056
6	32	111	390	1 404	7 917	19 383	76 461	331 387	1 253 615
7	50	168	672	2 756	11 988	52 768	249 660	1 223 050	6 007 230
8	57	253	1 100	5 060	39 672	131 137	734 820	4 243 100	24 897 161
9	74	585	1 550	8 268	75 893	279 616	1 697 688	12 123 288	65 866 350
10	91	650	2 286	13 140	134 690	583 083	4 293 452	27 997 191	201 038 922
11	104	715	3 200	19 500	156 864	1 001 268	7 442 328	72 933 102	600 380 000
12	133	786	4 680	29 470	359 772	1 999 500	15 924 326	158 158 875	1 506 252 500
13	162	851	6 560	40 260	531 440	3 322 080	29 927 790	249 155 760	3 077 200 700
14	183	916	8 200	57 837	816 294	6 200 460	55 913 932	600 123 780	7 041 746 081
15	187	1 215	11 712	76 518	1 417 248	8 599 986	90 001 236	1 171 998 164	10 012 349 898
16	200	1 600	14 640	132 496	1 771 560	14 882 658	140 559 416	2 025 125 476	12 951 451 931
17	224	1 610	19 040	133 144	3 217 872	18 495 162	220 990 700	3 372 648 954	15 317 070 720
18	307	1 620	23 800	171 828	4 022 340	26 515 120	323 037 476	5 768 971 167	16 659 077 632
19	338	1 638	23 970	221 676	4 024 707	39 123 116	501 001 000	8 855 580 344	18 155 097 232
20	381	1 958	34 952	281 820	8 947 848	55 625 185	762 374 779	12 951 451 931	78 186 205 824

# Voltage graph

## lift, quotient, assignment

- **Voltage graph** (large graph) can be obtained by **lifting** a **quotient** ( $= q$ , small graph) according to **voltage assignment** ( $= A$ )

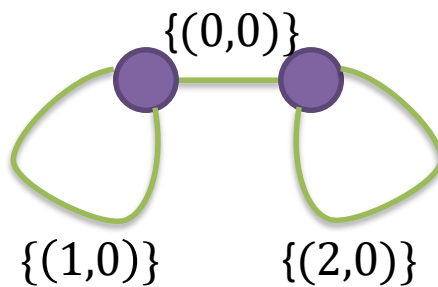
[Loz and Širáň, 2008]



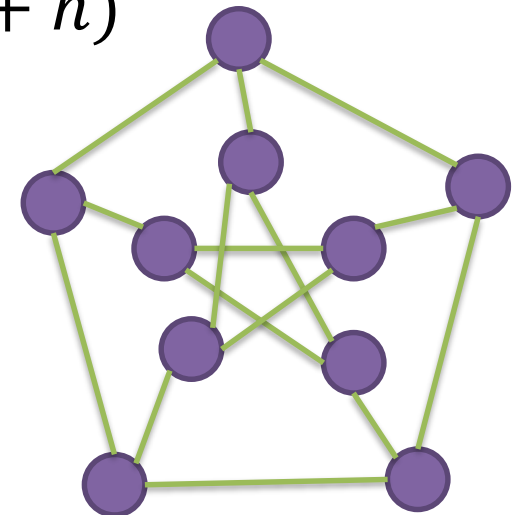
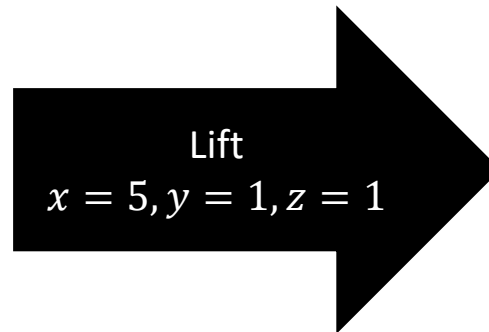
# Concrete example

- Use a map from  $E$  to the power set of  $Z_x \otimes Z_y$  as voltage assignment  $A$
- Use multiplication of **semidirect product**  $Z_x \rtimes_z Z_y$  defined as

$$(e, g) \times (f, h) = (e + z^g f, g + h)$$



$$D = (V, E)$$

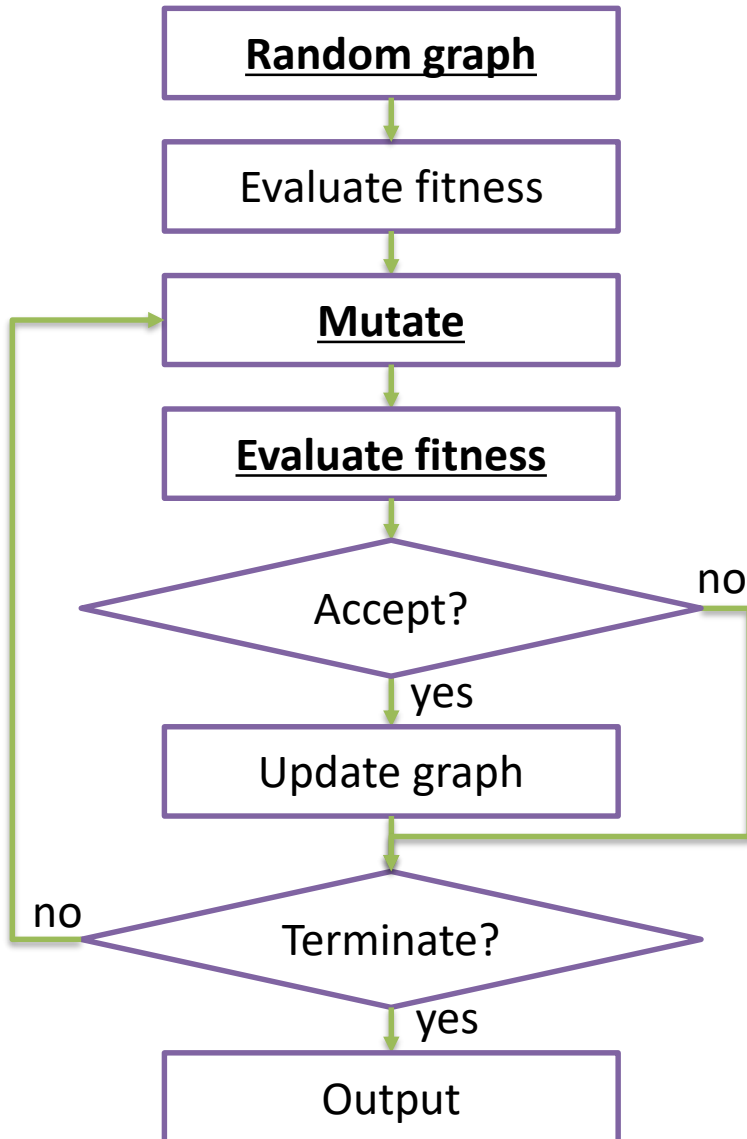


$$G' = (V', E')$$

$$V' = V \otimes (Z_x \otimes Z_y)$$

$$E' = \{((p, g), (q, h)) \mid (p, q) \in E, g, h \in (Z_x \otimes Z_y), h = g \times a, a \in A((p, q))\}$$

# Searching for a good voltage graph



## Random voltage graph

From given  $n$  and  $d$ , randomly choose  $q, A, w, x, y, z$  such that

$$q \in \{B, D, T, X\}$$

$$w := \begin{cases} 1 & \text{if } q = B \\ 2 & \text{if } q = D \\ 3 & \text{if } q = T \\ 4 & \text{if } q = X \end{cases}$$

$$x \times y \times w = n$$

$$1 \leq z \leq y$$

$$z^y \equiv 1 \pmod{x}$$

$A$  is random voltage assignment

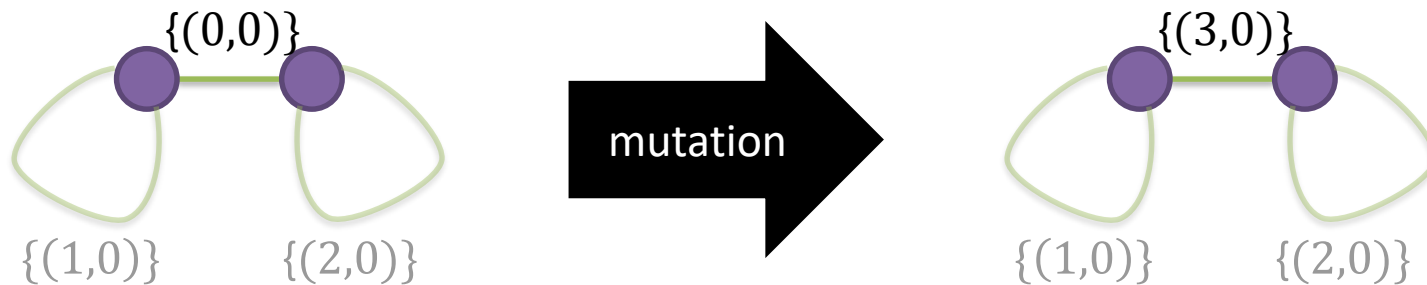
# Evaluation, mutation

## Evaluate fitness

Because of the symmetricity,  $w$  times of BFS is enough to compute  $k$  and  $ASPL$   
it becomes an  $O(n \times e \times w)$  time algorithm

## Mutation

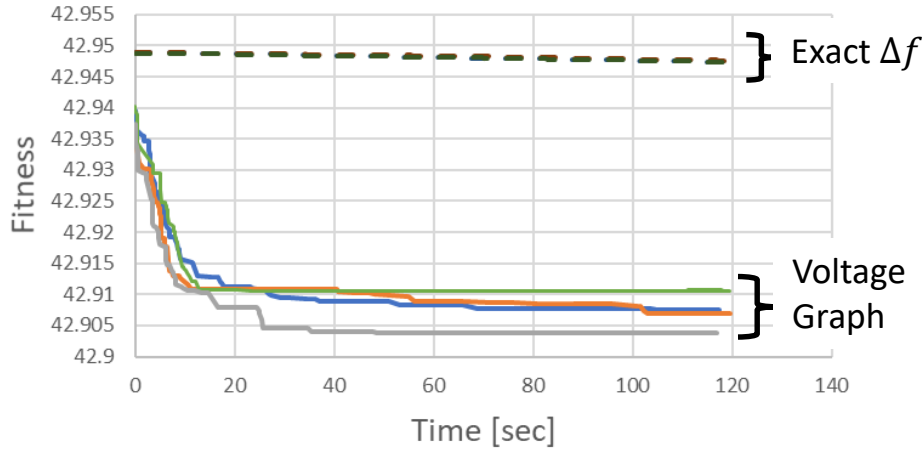
Randomly changing voltage assignment



# Local search of voltage graph

2 minutes of simulated annealing (4 times each)

$n=4896, d=24$



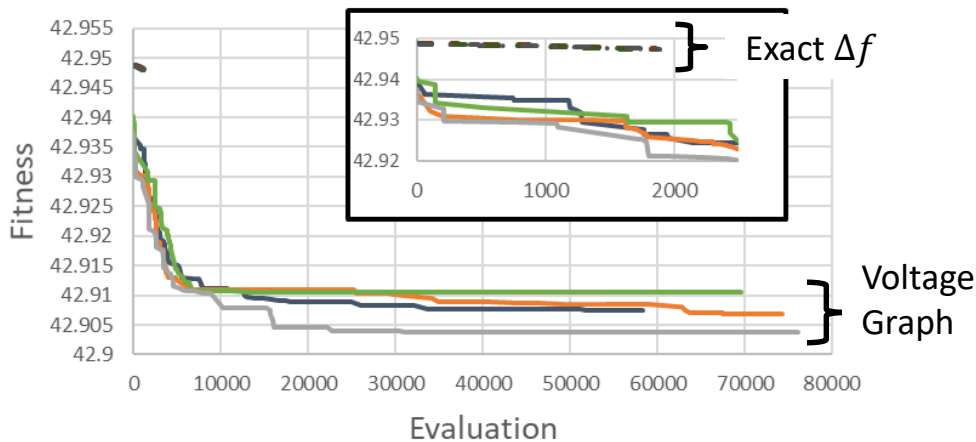
Relative improvement [%]

Exact  $\Delta f$       Voltage graph

$0.0033 \pm 0.0002$      $0.073 \pm 0.003$

22 times better

$n=4896, d=24$



Number of evaluation

Exact  $\Delta f$       Voltage graph

$1880 \pm 80$        $69600 \pm 7000$

37 times larger

# Result (general graph)

- Run 1~4 hours of simulated annealing algorithm using the voltage graph
  - Launch multiples times to change  $q, w, x, y, z$
  - Submit the best graphs for  $n = 100,000$  in Graph Golf 2017, and updated some of the records in Graph Golf 2016

	Order $n$	Degree $d$	Diameter $k$	$ASPL$	$ASPL$ (second best, last year best)
Graph Golf 2017	100,000	32	4	3.706	3.709
	100,000	64	4	3.015	3.016
Graph Golf 2016	1,024	8	5	3.500	3.505
	1,024	11	4	3.05	3.06
	1800	7	5	4.077	4.078
	10,000	7	6	5.411	7 (diameter)
	10,000	11	5	4.1064	4.1066
	10,000	20	4	3.375	3.376
	100,000	7	8	6.388	6.392
	100,000	11	6	5.14	5.15
	100,000	20	5	4.1326	4.1334



# Open questions, conclusion

- Why does the approximation usually work well?
- When does the approximation does not work well?
- How is the approximation related to other network analysis index such as edge betweenness?
- Is there any group other than semi-direct product to get good voltage graph?
- Is there other mathematical way to obtain a graph with good structure?
- Is there any better metaheuristic algorithm for the optimization by local search?
  
- Propose an algorithm to compute fitness value
  - $O(n^2 \times d)$  time for exact  $\Delta f$  and  $O(n \times d)$  time for approximate  $\Delta f$
  - $O(n \times d)$  memory
  - Use it for grid graph
- Employ a voltage graph
  - $O(n \times d)$  time for exact evaluation
  - $O(n \times d)$  memory
  - Use it for general graph
- Thank you! and question?